AFRL-RI-RS-TM-2011-001

FORMAL MODELS OF COMPOSABLE SECURITY ARCHITECTURES

OCTOBER 2011

TECHNICAL MEMORANDUM

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* OCTOBER 2011	**2. REPORT TYPE** INTERIM TECH MEMO	**3. DATES COVERED** *(From - To)* JAN 2009 – DEC 2010

4. TITLE AND SUBTITLE **FORMAL MODELS OF COMPOSABLE SECURITY ARCHITECTURES**	**5a. CONTRACT NUMBER** In-House
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) Dilia E. Rodriguez	**5d. PROJECT NUMBER** 23T4
	5e. TASK NUMBER PR
	5f. WORK UNIT NUMBER OJ
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITA 525 Brooks Road Rome, NY 13441-4505	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TM-2011-001

12. DISTRIBUTION AVAILABILITY STATEMENT
Distribution Approved for Public Release; Distribution Unlimited. PA# 88ABW-2011-5510
Date Cleared: 11 OCT 2011

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The objective of this project is to develop formal technology to support the development of secure systems. Presently much research and practice in security is concerned with particular enforcement mechanisms, and implementation or code-level vulnerabilities. At this late stage in the development of a system many security flaws are difficult to detect and fix. A more general formulation of security supports specification and analysis, and provides strong implementation-independent guarantees. It makes it possible to consider security from the early design of a system. To contribute toward this goal this project exploits results on the compositionality of information-flow properties to develop formal models and lightweight formal techniques that allow the specification and analysis of confidentiality and integrity requirements, and can be used to explore the design space of systems meeting such requirements.

15. SUBJECT TERMS
design of secure systems, information-flow security, lightweight formal methods

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** DILIA E. RODRIGUEZ
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	12	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

Contents

1 Introduction

The objective of this project is to develop formal technology to support the development of secure systems. Presently much research and practice in security is concerned with particular enforcement mechanisms, and implementation or code-level vulnerabilities. At this late stage in the development of a system many security flaws are difficult to detect and fix. A more general formulation of security supports specification and analysis, and provides strong implementation-independent guarantees. It makes it possible to consider security from the early design of a system. To contribute toward this goal this project exploits results on the compositionality of information-flow properties to develop formal models and lightweight formal techniques that allow the specification and analysis of confidentiality and integrity requirements, and can be used to explore the design space of systems meeting such requirements.

2 Technical Background

This research seeks to develop rigorous specification and analysis techniques to support the development of secure systems. The next section briefly describes their theoretical foundation. The following gives the preliminaries for the logic used to formalize them.

2.1 Information Flow and Basic Security Properties

Many security requirements can be met by restricting the flow of information. Goguen and Meseguer introduced this approach to security [3], and defined its fundamental property, *noninterference*. Intuitively, A is noninterfering with B if the actions of A have no effect on the observations of B. This provides confidentiality for A, and integrity for B. Over the years noninterference has been found to be either too restrictive, not restrictive enough, or not suitable for nondeterministic systems. As a result a number of variants of it have been defined, which have the same basic intuition, and collectively are known as information-flow properties. When they hold, many subtle attacks that exploit programs by using legitimate access to data are simply ruled out. Thus, this approach to security contributes to the goal of inherently secure systems.

To be able to manage the analysis of complex systems it is necessary to have some divide-and-conquer approaches. General theories of compositionality for safety and liveness properties, for example, make it possible to infer the properties of a system from the properties of its components. Information-flow properties, however, fall outside this framework. Furthermore, in general, these properties do not compose. However, over the years disparate treatments have shown that some properties, or restrictions of properties, do compose. [7, 8, 9, 10, 12]. More recently, Heiko Mantel proposed a unifying treatment of information-flow properties [4], in which these previous results can be rederived, and which also makes possible the definition of new information-flow properties, and the derivation of their compositional properties.

Mantel discovered a set of very basic elementary information-flow properties, which he calls Basic Security Properties (BSP's), and proved a collection of theorems stating how:

- some BSP's (some individually, others in combinations) imply other BSP's
- known information-flow properties are defined in terms of BSP's
- known information-flow properties are related by implication
- under what conditions these BSP's are preserved under composition
- under what conditions BSP's are trivially satisfied.

It turns out that under composition trivially satisfied BSP's might lead to the emergence of nontrivial information-flow properties.

With this framework as a theoretical foundation we developed an inference system for the specification and analysis of information-flow properties. The next section briefly describes the logic used.

2.2 Rewriting Logic and Maude

Rewriting logic [11, 1] is an executable computational logic that has been shown to be a very general logical and semantic framework [6, 5]. In rewriting logic computation and deduction coincide. It can be used to specify a system and define its semantics, or specify logics and formal tools that mechanize deduction.

A system or a logic is specified by a *rewrite theory*

$$R = (\Sigma, E, R),$$

where signature Σ defines its syntax, E is a set of equations, and R is a set of *rewrite rules*.

When $R = (\Sigma, E, R)$ specifies a system, the structure of its states is specified as an algebraic data type by (Σ, E), while its transitions are specified by the rewrite rules in R, which have the following form:

$$\boldsymbol{l} : \boldsymbol{t} \longrightarrow \boldsymbol{t'} \text{ if } cond.$$

A label $\boldsymbol{l}$ names each rule. The Σ-terms $\boldsymbol{t}$ and $\boldsymbol{t'}$ specify patterns of fragments of the state of the system. Whenever the condition *cond* holds, a local concurrent transition takes place: a state fragment that is an instance of $\boldsymbol{t}$ changes into a state fragment that is the corresponding instance of $\boldsymbol{t'}$.

Rewriting logic has no particular syntax, which makes it possible for (Σ, E) to define a domain-specific language, with whatever structure is appropriate. Consequently, the states and the transitions describe the system without any extraneous encodings imposed by the logic. This supports the goal of this research of lowering the technical barrier to rigorous specification and analysis of systems.

When $R = (\Sigma, E, R)$ specifies a logical system, Σ specifies its logical connectives; E, the structural properties of its propositions; and R, specifies the inference rules of the system, which concurrently rewrite formulas into other formulas, constructing proofs in the logic.

Rewriting logic has been implemented as the Maude system [2], which supports specification, programming, and verification. The models and techniques developed in this project have been implemented in Maude.

3 Compositionality Assistant

Component-based design provides a way of simplifying the design of complex systems. Components can be analyzed separately, and properties of systems composed from them can be inferred from those of the components. We have developed a *Compositionality Assistant*, a rewrite theory $R_{CA} = (\Sigma_{CA}, E_{CA}, R_{CA})$, implemented in Maude. Given some components assumed to satisfy some given security properties, it supports the exploration of designs of secure systems built from them.

3.1 Model

In various treatments of security, including Mantel's, a nondeterministic system is represented by an event system, a tuple (E, I, O, Tr), where E is a set of events, I and O are disjoint subsets of E, the set of input events, and the set of output events, respectively, and Tr is the set of behaviors of the system, which are sequences of events. Included in the signature Σ_{CA} are sorts (types) and operators to represent events and systems. The sets of events of a system would be represented by a term

```
|e> E |i> I |o> O
```

of sort Interface, where E, I, and O are terms of sort EventSet.

To define security requirements a set of security domains D is introduced, and for each domain $D \in D$, a view v_D. For example, a set of domains could be $\{H, L\}$, for High and Low, or in an agent-based application, different sets of agents could constitute the domains. A view for a domain D, $v_D = (V, N, C)$ partitions the set of events E into a set V of events that are visible from domain D, a set C of events that are confidential, kept secret from domain D, and a set N of events that are neither visible, nor secret, but could be deduced. We represent a view in Maude by a term

```
|v> V |n> N |c> C
```

of sort View, with subterms of sort EventSet.

To ensure that the constraints a view imposes on information flow are met a worst-case assumption is made: untrusted users might have complete knowledge of the system specification. BSP's are closure properties on the set Tr of traces of a system that require sufficiently many possible traces such that an adversary cannot deduce information of a particular kind [4]. Mantel's framework based on BSP's can express information-flow in the literature, such as separability, perfect security property, etc. These and new information-flow properties can be defined as conjunctions of BSP's. Maude supports object-based specifications, in which a configuration is constructed by an associative and commutative operator. Thus, we model a component that satisfies information-flow properties as a configuration of objects that denote the satisfaction of BSP's.

The formalization of the *Compositionality Assistant* has a class for each BSP. Each BSP is a closure property on Tr. If there is some trace $t \in Tr$ of a particular form, described in terms of elements of the sets of the view, and for some BSP's, of other subsets of E, it requires that some trace t' obtained from t by the deletion or insertion of some specified events, be also in Tr. To express that a component specified by the event system $ES = (E, I, O, Tr)$, for a given view $v_D = (V, N, C)$, satisfies a BSP, say BSD, we use an object of the following form:

```
< Id  : BSD |
    interface : |e> E |i> I |o> O,
    view      : |v> V |n> N |c> C > .
```

Classes for other BSP's have additional attributes for the parametric sets of events they require.

Each component has an identifier, above Id. A secure component, that is, one that satisfies one or more BSP's, or properties defined in terms of BSP's, is modelled by a configuration of objects, each with the component identifier.

3.2 Design-Space Exploration

Theorems in [4] relate BSP's of a component by implication, define under what conditions BSP's of components are preserved when they are composed, and conditions in which BSP's are trivially satisfied. These theorems are formalized in R_{CA} as rules of the form:

$$l : t \rightarrow t' \quad \textbf{if} \ cond,$$

where t and t' are configurations of objects denoting security properties of components. If the condition *cond* is true the security property represented by t implies the security property represented by t'. Proofs are constructed by the application of these rules. Given a configuration of objects specifying the properties a set of components satisfies it is possible, in general, to construct proofs of other properties the components satisfy, and by equationally "connecting" outputs of one component to inputs of another, to construct proofs of properties of the resulting composite system.

Aside from the BSP classes mentioned above, there are also superclasses BasicSecProp and SecurityProp.

```
sorts Total R BSD BSI BSIA FCD FCD BasicSecProp SecurityProp .
subsort Total < SecurityProp .
subsorts R BSD BSI BSIA FCD FCI < BasicSecProp < SecurityProp .
```

Given a configuration expressing the security properties of a set of components, it might be possible to construct zero, one, or many properties implied by those denoted by the configuration.

Using Maude's search command we may query the system about security properties of interest. This would automatically generate proofs by the application of the rules in R_{CA}, and present the one or more solutions requested, or report that no solution to our query was found.

For example, suppose we have two components, with identifiers id1 and id2. A configuration of objects would specify their security properties, and equations identifying outputs of one to inputs of the other would specify the particular composition of these components to be analyzed. The inference system deduces properties implied by those of the components, in the specified composition, by transforming this initial configuration. The security properties of any component or system are represented by objects with the identifier of the system. A system composed from components with identifiers id1 and id2 would have identifier id1 . id2. Thus, the presence of an object with such identifier would indicate that the corresponding composite system satisfies a security property.

To learn, for example, whether the composite system satisfied any BSP we would submit the following query:

```
search system
  => *
  [ < id1 . id2 : BasicSecProp | Atts:AttributeSet >
    C:Configuration ]                                    .
```

Given a specification system, this command queries whether the inference system can construct a configuration with the pattern described within the square brackets. Such a configuration would have an object with identifier id1 . id2 of the class BasicSecProp, which is a superclass of each BSP-class. Neither the particular values of the attributes of this object, nor the particular constituents of the rest of the configuration are of interest, and so these are represented by variable Atts of sort AttributeSet and variable C of sort Configuration, respectively. Any configuration with this pattern represents the statement that the composite system with identifier id1 . id2 satisfies some BSP. This query would generate all the possible proofs of this statement, and return all the configurations matching the pattern.

To learn whether there is some particular BSP, say *BSD*, that this system satisfies the query would be:

```
search [1] system
  => *
  [ < id1 . id2 : BSD | Atts:AttributeSet >
    C:Configuration ]                                    .
```

Only one solution would be generated, if there is at least one proof. (We could specify some other limit on the number of solutions.) Otherwise, the system would report that there is no solution.

The views of the components induce a set of possible views for systems composed from them. If interested in a particular view, the query would have the following form:

```
search [1] system
  => *
  [ < id1 . id2 : BSD |
        view : |v> V:EventSet |n> N:EventSet |c> c12,
        Atts:AttributeSet >
    C:Configuration ]                                   .
```

The subterms of the view could all be ground (having no variables), or we could constrain only one of the sets of the view. The above example, is concerned with a view that keeps the set of events c12 secret. It does not matter whether the remaining events are visible or deducible. Only if *BSD* is satisfied for this particular specification view, would there be a solution.

One may also define information-flow properties known from the literature, or newly defined for a particular application. A new class would be introduced:

```
sort MyInfoFlowProp .
subsort MyInfoFlowProp < SecurityProp .
```

Included in the set of equations E_{CA} there would be a configuration of BSP objects defined to be our new property. The following query would determine whether a composite system could satisfy it.

```
search [1] system
  => *
  [ < id1 . id2 : MyInfoFlowProp |
        view : |v> V:EventSet |n> N:EventSet |c> c12,
        Atts:AttributeSet >
    C:Configuration ]                                   .
```

With these sorts of queries, and also with model-checking commands, it is possible to explore the design space for some application or system.

Securing cyberspace is a major concern for government, military, and commercial enterprises. Much of the practice and research on security addresses it at the implementation or late stages of system development. Other approaches are based on semiformal methods, which are not suited for the rigorous analysis that the security of systems requires. Security needs to be considered from the earliest stages of system design, and for this rigorous, formal support is required. This project has sought to contribute toward that end by taking as its theoretical foundation the very flexible compositionality results from Mantel's information-flow security framework, and by developing lightweight formal methods that exploit those results to support the exploration of the design space. We briefly indicate below the objective of the remainder for this project.

4 Next

While the system we have described can be useful in designing secure systems, it can have spatial and time limitations. BSP's for one view and parametric event sets induce BSP's for other related views and subsets of the parametric sets. Thus, we have a combinatorial growth in the proof search space. While we developed some optimization techniques that reduced this search space, we will investigate next models that would have a greater impact on the efficiency and scalability of our techniques.

References

[1] Roberto Bruni and José Meseguer. Generalized rewrite theories. In *Proceedings of ICALP'03*, in Lecture Notes in Computer Science, vol. 2719, 2003, pp. 252–266.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude — A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. Springer, 2007.

[3] Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy*, pp 11–20, 1982.

[4] Heiko Mantel. On the composition of secure system. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–101, May 2002.

[5] Narciso Martí-Oliet and José Meseguer. General logics and logical frameworks. In: Gabbay, D. M. (ed.) What is a Logical System? Studies in Logic and Computation, vol. 4, pp. 355–392. Oxford University Press, Oxford (1994)

[6] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In: Gabbay, D.M., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 9, 2nd. edn., pp. 1–87. Kluwer Academic Publishers, Dordrecht (2002)

[7] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *IEEE Symposium on Security and Privacy*, pp 161–166, 1987.

[8] D. McCullough. A Hookup Theorem for Multileve Security. *IEEE Transactions on Software Engineering*. 16(6), 1990.

[9] J. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *IEEE Symposium on Research in Security and Privacy*, pages 79–93, 1994

[10] J. McLean. A General Theory of Composition for a Calss of ”Possibilistic” Security Properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[11] José Meseguer. Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96(1) (1992) pp. 73–155.

[12] A. Zakinthinos and E. S. Lee. A General Theory of Security Properties. In *IEEE Symposium on Security and Privacy*, pages 94–102, 1997.